Курс: Функциональное программирование
Практика 10. Трансформеры монад

Разминка

► Устно вычислите значения выражений и проверьте результат в GHCi:

```
msum [Just 1,Just 2,Just 3]

msum [Nothing,Nothing,Just 1,Just 2]

msum [[1,2,3],[10,20]]

mfilter (<10) $ Just 12

mfilter (<10) $ [1,3..]

do {x <- Just 5; guard (x<10)}
```

Монада Error

Введём тип данных для представления ошибки синтаксического разбора

```
data ParseError = Err {location::Int, reason::String}
```

Сделаем этот тип представителем класса `Error`

```
instance Error ParseError where
  noMsg    = Err 0 "Parse Error"
  strMsg s = Err 0 s
```

Объявим нашу собственную `Error`-монаду

```
type ParseMonad = Either ParseError
```

► Почему `ParseMonad` является монадой?

► Разработайте функции

```
parseHex :: String -> ParseMonad Integer
printError :: ParseError -> ParseMonad String
```

`parseHex` пытается разобрать переданную ей строку как шестнадцатиричное число. При удачном исходе она возвращает это число, а при неудачном — генерирует исключение. `printError` выводит информацию об этом исключении в удобном текстовом виде. Для тестирования используйте

```
test s = let (Right str) = do {n <- parseHex s; return $ show n} `catchError` printError
         in str
```

Совет: воспользуйтесь вспомогательными функциями из `Data.Char`.

Трансформеры монад

▶ Разберитесь в работе следующего кода:

```
askPassword :: MaybeT IO ()
askPassword = do
  liftIO $ putStrLn "Insert your new password:"
  value <- msum $ repeat getValidPassword
  liftIO $ putStrLn "Storing in database..."

getValidPassword :: MaybeT IO String
getValidPassword = do
  s <- liftIO getLine
  guard (isValid s)
  return s

isValid :: String -> Bool
isValid s = length s >= 8
            && any isAlpha s
            && any isNumber s
            && any isPunctuation s
```

▶ Модифицируйте этот код так, чтобы он выдавал пользователю сообщение о причине, по которой пароль отвергнут.